Video Capture Documentation

Release 0.0.0.1

roxlu

May 01, 2017

Contents

Video Capture is a cross platform library to capture video frames from capture devices. This library uses modern SDKs/APIs for Mac, Linux and Windows. This library is tested and developed on Win 8.1, Mac 10.9 and Arch Linux (latest Linux).

Contents:

CHAPTER 1

Getting Started

To compile Video Capture you need to do:

- Make sure that you installed all dependencies
- Clone the Video Capture repository from github
- Compile using the build script

Building the library

Video Capture primary location is Github. To get the code clone the project:

```
git clone git@github.com:roxlu/video_capture.git
```

Dependencies

Video Capture main development systems are Mac OS 10.9, Windows 8.1 and Arch Linux. On Linux we use the Video4Linux API, on Mac we use AVFoundation which are both part of the OS. On Windows you need to download the latest Windows SDK which provides the MediaFoundation libraries. We use CMake to compile the library and examples. The Video Capture library contains an OpenGL example. For this OpenGL example we depend on libglfw 3.

Compiling Video Capture on Mac and Linux

For Mac and Linux systems we use the same compile script. To compile follow these steps.

```
cd build
./release.sh
```

Compiling Video Capture on Windows

On windows we use CMake too with a build script. Development uses Microsoft Visual Studio 2012 Express. To compile on Windows follow these steps.

```
cd build
build.bat 64 release
```

Compiling programs that use Video Capture

To compile a program that uses Video Capture make sure to link with the created *libvideocapture.a* on Mac and Linux and the *libvideocapture.lib* file on Windows. The library is installed in the *install* directory that we create when you use the above describe build steps.

Also make sure to add a header search path to the headers that we also install into the *install* directory.

Libraries to link with on Linux

- udev

Libraries to link with on Mac

- CoreFoundation Framework
- AVFoundation framework
- Cocoa
- CoreVideo
- CoreMedia

Libraries to link with on Windows

- Mfplat.lib
- Mf.lib
- Mfuuid.lib
- Mfreadwrite.lib
- Shlwapi.lib

CHAPTER 2

Programmers Guide

In this guide we will create a very simple program that lists the available capture devices, then lists the capabilities of a device. Once we found a capability that we want to use we open the device and start captureing. Then in a loop we will flush the buffers of the capture device and let it call our callback function. Before we start we explain a couple of concepts that we use in Video Capture.

Concepts used in Video Capture

In Video Capture we use a couple of concepts which are shared among most SDKs/APIs we found on OSX, Linux and Windows.

Device We use the term **Device** to represent something like a webcam. This **Device** can capture video in a specific **pixel format**.

Pixel format A pixel format describes how the bytes of a video frame are stored. Common pixel formats for Video Capture are YUYV422, UYVV422 and YUV420P. Some OSes can convert between pixel formats (Mac). See [libyuvs](https://code.google.com/p/libyuv/wiki/Formats) documentation for some more info on format mappings..

Output formats Some SDKs have optimized solutions to decode a video stream you get from a capture device. For example on Mac you can use the OS to convert a raw YUV stream into a RGB24 stream. Some OSes even have support to decode H264, so the output formats are also related to codecs and not only pixel formats.

Capability A capability describes a couple of things related to what a device can give you. These are things like the dimensions of the video frames you receive, the framerate and the pixel format. Video Capture supports querying the available capabilities of a device on Windows, Mac and Linux.

Settings Video Capture uses a settings object when you want to open a device. The Settings object stores information like, what device you want to use, what capability and what pixel format you want to use. A settings object is passed into the `open()` method of the cature class.

Frame A frame is a helper type we created which gives you information about a pixel format. If necessary you can use a `Frame` object to get information about strides, widths, heights, offsets etc.. for planar or non-planar pixel formats. See the opengl example where we use a Frame to get offets into the YUV420P data on windows.

Getting information about a device

Before we open a capture device we have to inspect the capabilities of the capture device. Do detect if we found your capture device you can use the `listDevices()` function of the `Capture` class.

Note that all types of the Video Capture library are using the `ca` namespace. The example below creates a `Capture` instance which is the interface to your capture device.

```
using namespace ca;

Capture capture(fc,NULL);
if(capture.listDevices() < 0) {
  printf("Error: cannot list devices.\n");
  ::exit(EXIT_FAILURE);
}

if(capture.listCapabilities(0) < 0) {
  printf("Error: cannot list capabilities for devices 0.\n");
  ::exit(EXIT_FAILURE);
}
```

`listDevices()` will log all the found capture devices to stdout. Each device has a unique number that you will need to use when opening a device. When a function fails it will return a negative error code, so make sure to check if the result is `< 0` as shown above.

The `listCapabilities()` function will list all the capabilities of the capture device. A capability describes the width, height, framerate and pixel format. From this list, pick the capability number that you want to use. The Video Capture library also provides a `findCapability()` function that you can use to find a specific capability for a device. This function will return the index number of the found capability or a negative value if not found.

Some SDKs can convert a pixel format from the capture device into another, maybe more easy to use one. For example Mac gives you a way to convert from a YUV pixel format to RGB format. Although this is very handy, it's not recommended because converting will mostly be done on the CPU (maybe with SIMD) which means you loose some processing power for other parts of your application. Use the `listOutputFormats()` to inspect what output formats are supported.

Opening a device

Once you've found what *device*, *capability* and *output format* you want to use you need to create a `Settings` object that describes how you want to use the device. Below we show how to create a `Settings` object and how to set what capability, device and format we want to use.

```
using namespace ca;

Settings settings;
settings.device = 0;         // Use number 0 from the device list (see listDevices())
settings.capability = 15;    // Use number 15 from the capability list (see␣
↪listCapabilities())
settings.format = -1         // We're not using any output format conversion (see␣
↪listOutputFormats())
```

Once we have this `Settings` object we pass it into the `open()` function of the `Capture` instance. This will open the device and set the capability. Make sure to check the return values from `open()`, when it's negative an error occured.

```
using namespace ca;
Capture capture(fc, NULL);

Settings settings;
settings.device = 0;
settings.capability = 15;
settings.format = -1

if(capture.open(settings) < 0)) {
  printf("Error: cannot open the capture device.\n");
  :exit(EXIT_FAILURE);
}
```

Captureing frames

After opening the capture device we can start receiving frames. Video Capture uses a callback function that is called whenever a new frame arrives. Two things are important about this callback function:

- This function may be called from another thread
- This function must return before a new frame arrives

The callback function is passed to the constructor of the `Capture` class. The interface of this callback function is:

```
void on_frame(void* bytes, int nbytes, void* user)
```

- **bytes** is a pointer to the frame data from the capture device. This maybe be a pointer to planar video data when e.g. using YUV420P. See the opengl example where we use YUV420P (on Windows).
- **nbytes** the number of bytes in the frame.
- **user** a pointer to user data. This is the second parameter to the `Capture` constructor.

To capture frames you use:

```
if(capture.start() < 0) {
   printf("Error: cannot start captureing.\n");
   ::exit(EXIT_FAILURE);
}

while(must_capture) {
   capture.update();
}

if(capture.stop() < 0) {
  printf("Error: cannot stop the capture process.\n");
}
```

Make sure to call `update()` at at least the same rate of the used frame rate. Some capture SDKs don't use async callbacks for which we need to process any pending frames.

Closing a device

Once you're done make sure to correctly cleanup and shutdown the capture device. Closing a device will make sure that all allocated memory gets freed and the device is correctly shutdown. Note: when you don't close a device on

Linux, it will continue to be in *opened* state and can't be used anymore, before you correctly close it.

```
if(capture.close() < 0) {
   printf("Error: cannot close the capture device.\n");
}
```